Introduction to FFAST in Digital Multitone Communication

Digital communication is a pervasive technology that many individuals, corporations, and governments rely upon to share information. Numerous researchers and engineers have focused their efforts on decreasing the time required for digital communication. One of the greatest such advancements is the Fast Fourier Transform (FFT), an algorithm [link] developed by James Cooley and John Tukey in 1965 that reduces the computational complexity of the DFT from $O(\ ^2)$ to $O(\ \log\ )$ for appropriately chosen , where is the signal length.

For many years, $O(\ \log\ )$ was believed to be the best achievable run-time complexity for computing a discrete time Fourier transform (DFT). However, in recent years, researchers have developed a class of algorithms known as Sparse FFT algorithms [link] which run in sublinear time with respect to . These algorithms are able to have such a low run time complexity because they assume that the signal is *sparse* in its Fourier representation; that is, there are nonzero DFT coefficients where $<<$.

In this report, we present one such Sparse FFT algorithm, the Fast Fourier Aliasing-based Sparse Transform (FFAST)[link], as a method to significantly decrease computation time in a digital multitone (DMT) communication scheme because it can operate in $O(\ \log\ )$. In Module 2, we present the DMT scheme and demonstrate signal sparsity. In Module 3, we outline the architecture of FFAST. In Module 4, we highlight several FFT methods used in our project. We describe our computational experiment and provide numerical results in Module 5 and discuss practical considerations and other future work in Module 6.

Digital Multitone Modulation

Digital Multitone Modulation

We present the digital multitone modulation scheme and demonstrate its suitability for demodulation via FFAST.

Digital Multitone Modulation Scheme

Let a finite alphabet be given, where each symbol is associated with a unique sequence of ordered bits, , where and for . For example, let be the set of all lowercase letters in the English alphabet and associate each letter with its order in the alphabet. In this case, the binary sequence `01101' corresponds to the thirteenth letter, “m".

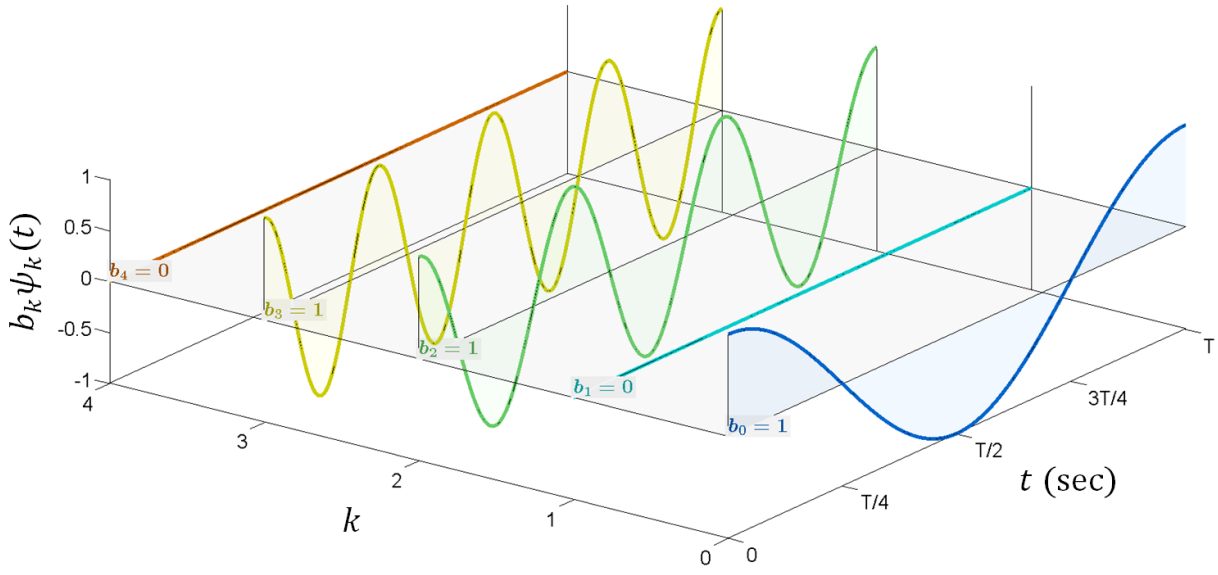
Generally speaking, Digital Multitone (DMT) Modulation is a “parallel communication scheme in which several carriers of different frequencies each carry narrowband signals simultaneously” [link]. These narrowband signals are usually sinusoids that encode the binary sequence associated with each symbol. If a bit is “high", then the corresponding sinusoid is expressed in the output signal; otherwise the bit is “low" and the sinusoid is not expressed. More precisely, given a symbol with the corresponding binary sequence , the message signal is defined to be

Equation:

---

for some fundamental carrier frequency, .

In our previous example where is the English alphabet and the letter “m" corresponds to “01101”, the message signal is the sum of the first, third, and fourth harmonics, as shown in the figure below:

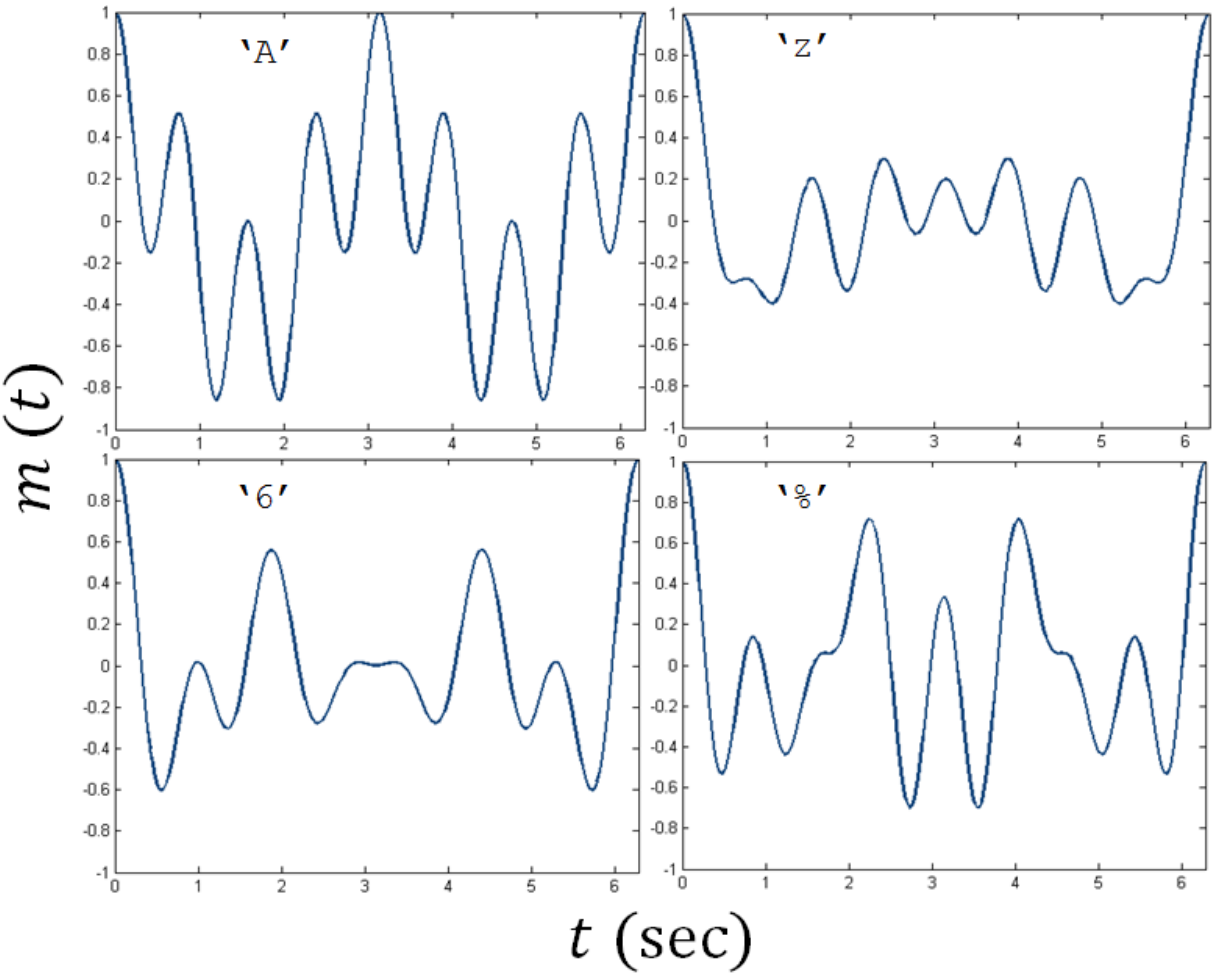


Decomposition of “m” in DMT Scheme

In our computational experiments, we use digital multitone modulation to encode 8-bit Extended ASCII values. An Extended ASCII table can be found here. Below are several symbols and their digital multitone modulation signals.

Char	Hex	Binary
A	41	0100 0001
z	7A	0111 1001
6	36	0011 0110
%	25	0010 0101

Table of Extended ASCII Values



Different Symbols in DMT Scheme

Sparsity in Digital Multitone Modulation

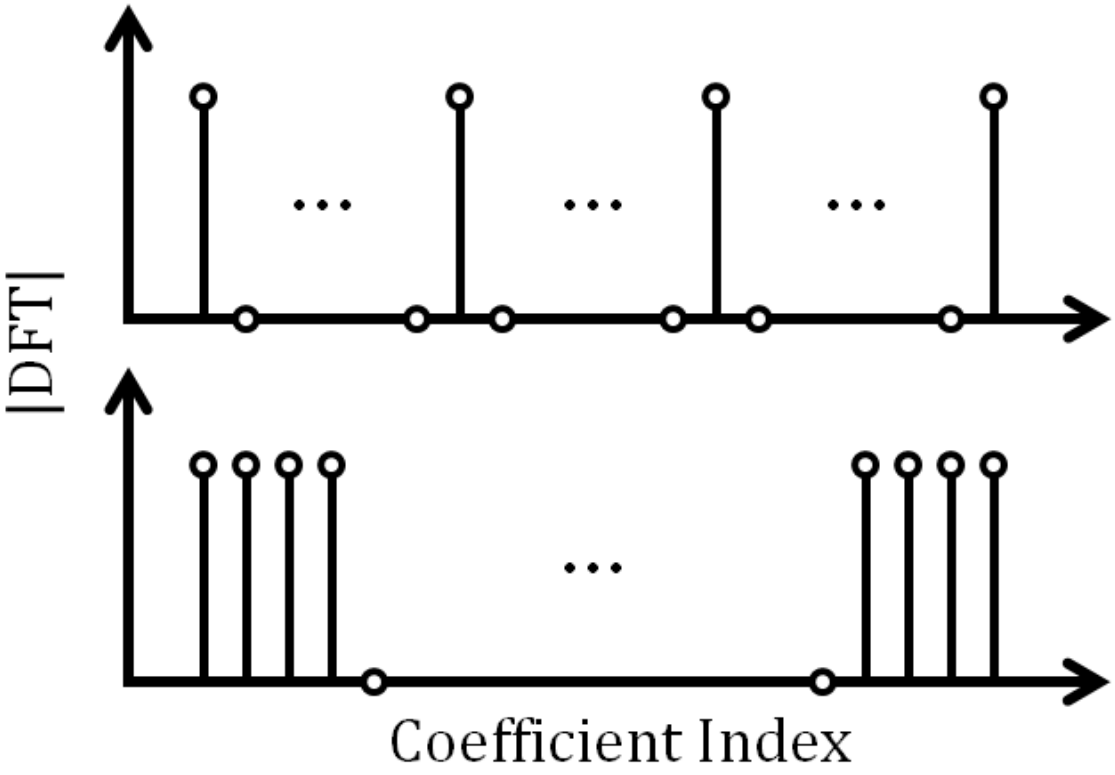
Sparse FFT algorithms only achieve low runtime complexity if the input signal is sparse in its Fourier representation; that is, if for a length- signal, there are nonzero DFT coefficients with . FFAST, the sparse FFT algorithm that we will be using, requires the sparsity constraint ¯. Recall that the message signal, , is defined as

Equation:

so that the Nyquist frequency is . In order to ensure signal sparsity, the sampling frequency should be a multiple of the fundamental carrier frequency so that each of the sinusoidal components falls into a single frequency bin. This type of “on-the-grid" sampling may be expressed as
Equation:

where is the length of the sampled signal. Note that in [link] each sinusoid contributes two DFT coefficients. Thus, if all bits are high so that the sparsity condition may be expressed as ¯.

Sampling plays a large role in signal sparsification. There are many sampling methods that ensure sparsity and we present two different methods. The first method involves padding the input signal to achieve sparsity. Consider sampling at the Nyquist frequency so that . As stated, this sampled signal is not necessarily sparse – in fact, if all bits are high! However, periodizing the sampled signal sufficiently many times will result in higher frequency resolution by placing zero-value coefficients in between the nonzero coefficients, thus sparsifying the signal. This method results in a spectrum with nonzero coefficient few and far between. Second, consider sampling at a sufficiently high rate to satisfy the sparsity condition; that is, . First note that so that aliasing does not occur. This method results in a compact spectrum where only the first and last coefficients are nonzero. See the figure below for a spectra that are characteristic of these methods.



Spectra for Different Sampling Schemes

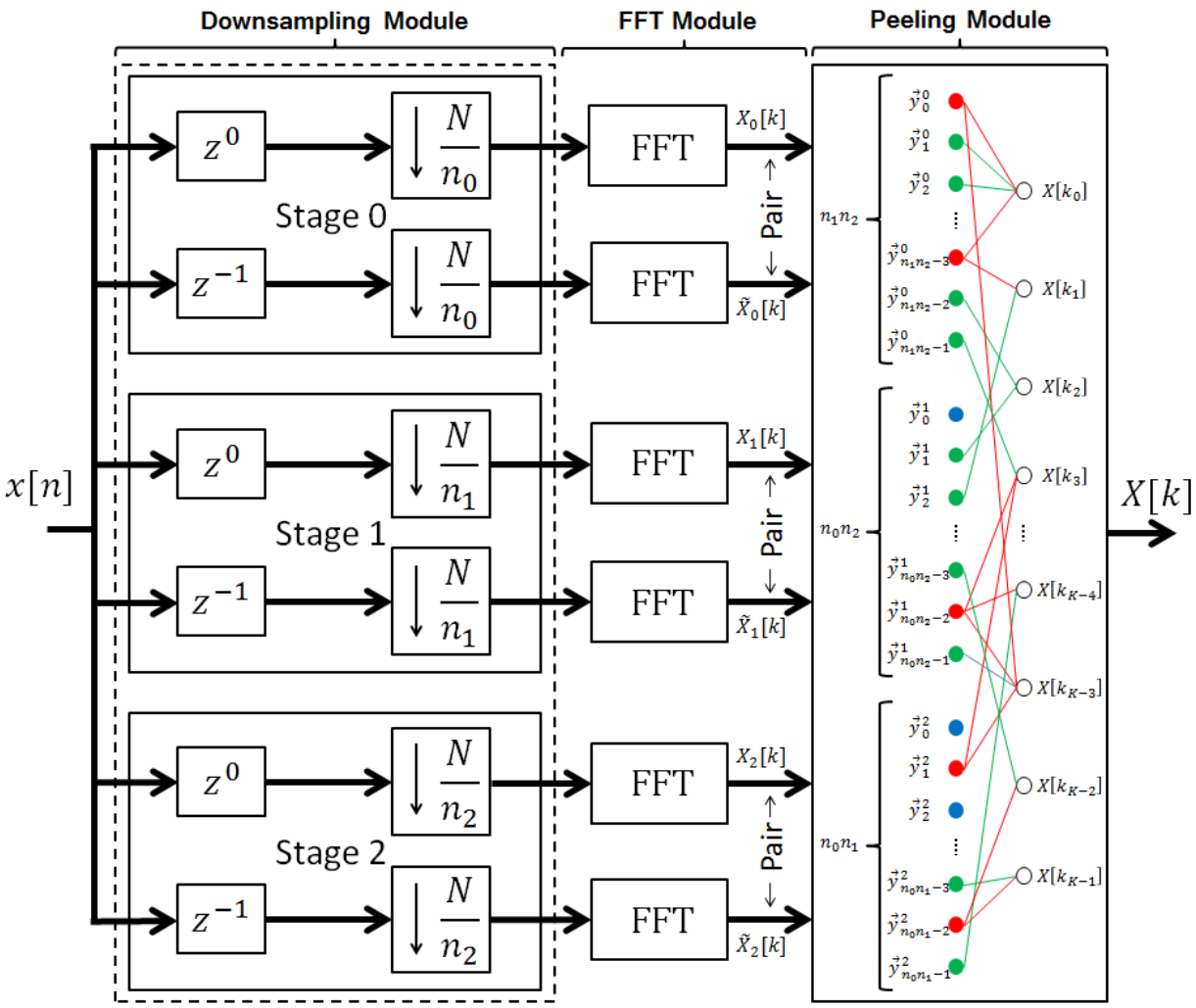
It should be noted that there are many sampling methods that ensure signal sparsity but for the purposes of this project, we care only that the signal is sparse. Sampling schemes are discussed further in [link].

Fast Fourier Aliasing-based Sparse Transform
A brief description of the FFAST algorithm.

Fast Fourier Aliasing-based Sparse Transform

Fast Fourier Aliasing-based Sparse Transform (FFAST) is a sparse FFT algorithm developed by Sameer Pawar and Kannan Ramchandran in May 2013 [link]. We present a formulation of the algorithm that is specialized for our digital multitone scheme.

FFAST consists of three modules: the downsampling module, the FFT module, and the peeling module. See [link] for a diagram of this architecture.



FFAST Architecture

Front End

The downsampling module consists of three stages. In each stage i, the signal and a delayed version are both downsampled by a sampling coefficient, $n_i \in \mathbb{Z}_+$. This introduces aliasing in the frequency domain, which will be a key component in the peeling module. It is necessary that the sampling coefficients n_i are coprime factors of the singal length N; that is, $n_0\, n_1\, n_2 = N$ where n_0, n_1, n_2, are all relatively prime.

These smaller subsignals are passed to the FFT module, which computes the DFT of each subsignal. Any FFT algorithm may be used in this stage with the condition that it works for a general signal length N. The DFTs of the subsignals at stage i are then paired together. We denote such a pair as $\overrightarrow{y_l^i} = (x_i[l], \widetilde{x}_i[l])$, where $x_i[l]$ and $\widetilde{x}_i[l]$ are the l_{th} values of the DFT of the normal and delayed subsignals of stage i, respectively.

Back End

The peeling module takes these smaller DFT pairs and backsolves a bipartite graph to obtain the DFT coefficients of the original signal. To understand the structure of this graph, recall that the aliasing caused by downsampling “mixes" frequency domain components. More precisely, the coefficients of the smaller DFTs are a linear combination of the original DFT coefficients. Consider a graph with two types of vertices: the smaller DFT pair coefficients $\overrightarrow{y_l^i}$ and original DFT coefficients $X[p]$. If an original DFT coefficient contributes to the value of a smaller DFT coefficient, an edge is placed between the two vertices. It is easy to see that this is a bipartite graph because the vertex set can be partitioned into smaller DFT coefficients and original DFT coefficients.

We denote a smaller DFT coefficient vertex as a *zero-ton* if no nodes are connected to it, a *singleton* if exactly one node is connected to it, and a *multi-ton* if it is neither a zero-ton nor a singleton.

If a vertex $\overrightarrow{y_l^i} = (x_i[l], \widetilde{x}_i[l])$ is a pair of zeros, then it is a zero-ton. Otherwise, to determine whether a vertex is a zero-ton, a singleton, or a multi-ton, the algorithm uses a “Ratio Test" [link]. Recall that a circular shift in the time domain is a multiplication by a complex exponential in the frequency domain so that we may use the values in $\overrightarrow{y_l^i}$ to determine whether the vertex is a singleton. To perform this ratio test we may check if the quantity

Equation:

$$q = \frac{N}{2\pi} \quad \angle \frac{\widetilde{x}_i[l]}{x_i[l]}$$

is an integer. If q is an integer, then the vertex in question is a singleton and thus, $X[q] = x_i[l]$; otherwise, the vertex in question is a multi-ton.

We now describe the process of backsolving this bipartite graph to get the DFT coefficients of the original signal. If a vertex is a zero-ton, we may remove it from the graph because it provides no relevant information. If a vertex is a singleton, we have obtained a DFT coefficient $X[q]$. By the "mixing" process of aliasing, we know which smaller DFT pairs $\overrightarrow{y_l^i}$ that $X[q]$ contributes to. With this information, we may subtract $X[q]$ from these smaller DFT pairs, thus removing edges from the graph. We repeat these steps until all edges are removed from the graph and $X[q]$ is known completely. This process is known as *peeling* and is reminiscent of decoding Low Density Parity Check codes.

Convergence Conditions

In general, FFAST is a robust algorithm that can handle noise, many signal lengths N, and sparsity factors k, where k is the number of nonzero DFT coefficients. We presented a specific noiseless version of the algorithm that requires the sparsity constraint $k < N^{1/3}$. As previously mentioned, FFAST also requires that the subsampling coefficients are coprime factors of N. With these conditions, FFAST is guaranteed to converge to a solution almost surely.

FFT Methods

Self-Sorting Mixed-Radix FFT

The mixed-radix approach utilizes clever subsampling of $x[n]$ and permutations of twiddle factor matrices to lower operation counts. The first step is to compute the prime factorization of the signal length N. Prime factors of 2 and 3 are then combined to create as many factors of 4 and 6 as possible. The resulting prime factorization has the form $N = \prod_{i=1}^{F} f_i$. We can then perform a single step T_i of the algorithm by computing N/f_i FFTs of length f_i.

Four unique matrix constructions are necessary to generalize a single step of the algorithm. The first is the identity matrix $I_r \in \mathbb{R}^{r \times r}$. The second is the permutation matrix $P_b^a \in \mathbb{R}^{ab \times ab}$, where
Equation:

$$P_b^a\left(j,k\right) = \begin{cases} 1 & \text{if } j = ra + s \text{ and } k = sb + r \\ 0 & \text{otherwise} \end{cases}$$

Note that P_b^a swaps positions $ra + s$ and $sb + r$ in a vector. The third matrix to consider is a diagonal matrix of twiddle factors $D_b^a \in \mathbb{R}^{ab \times ab}$, where
Equation:

$$D_b^a\left(j,k\right) = \begin{cases} \omega_{ab}^{sr} & \text{if } j = k = sb + r \\ 0 & \text{otherwise} \end{cases}$$

with $\omega_{ab} = e^{-j2\pi/(ab)}$. The fourth and final matrix construction to consider is the standard DFT matrix $W_n \in \mathbb{R}^{n \times n}$.

The mixed-radix algorithm requires the interaction of operations on the subspaces $\mathbb{R}^i,\ i = 1, \ldots, F$, and so it is necessary to consider the Kronecker Product in calculations. The Kronecker Product is an operation defined as $\otimes : \mathbb{R}^{m \times n} \times \mathbb{R}^{p \times q} \to \mathbb{R}^{mp \times nq}$. If $C = A \otimes B$, then
Equation:

$$C\left(v, w\right) = A\left(i, j\right) B\left(k, l\right) \quad \text{and} \begin{cases} v \equiv p(i-1) + k \\ w \equiv q(j-1) + l \end{cases}$$

Using these matrices, it possible to compose a single steps T_i using the following equation [link]:
Equation:

$$T_i = \left(P_{q_i}^{f_i} D_{q_i}^{f_i} \otimes I_{p_{i-1}}\right) \left(W_{f_i} \otimes I_{m_i}\right)$$

where
Equation:

$$\begin{aligned} p_i = \textstyle\prod_{j=1}^{i} f_j \quad (p_0 = 1), &\quad i = 0, \ldots, F \\ q_i = N / p_i, &\quad i = 1, \ldots, F \\ m_i = N / f_i, &\quad i = 1, \ldots, F \end{aligned}$$

Note that $T_i \in \mathbb{R}^{N \times N}$.

To implement the algorithm, each stage is applied iteratively to acquire X. T_i may be simplified into two separate steps using the definitions of each matrix to yield the following algorithm structure:

```
for i = 1:nf
    for a = 0:(q(i+1)-1)
        for b = 0:(p(i)-1)
            xsub = x(b+(0:f(i)-1)*m(i)+a*p(i) + 1);
            if f(i) <= 6
                t = WinogradFFT(xsub);
            else
                t = RaderFFT(xsub);
            end

            for lambda = 0:(f(i)-1)
                xprime((a*f(i)+lambda)*p(i)+b + 1) = ...
                    exp(-2*pi*1i/q(i)*lambda*a)*t(lambda+1);
            end
        end
    end
    x = xprime.';
end
xk = x;
```

Mixed Radix

The DFT module in the above code utilizes the Winograd Short-Length Transforms to minimize operations when computing DFTs of length less than 6. For all other lengths, Rader's FFT is used.

Rader's FFT

Rader's FFT for prime lengths exploits results from number theory to express the DFT as a composite-length cyclic convolution [link]. Any prime number p defines a multiplicative group modulo p, denoted $\mathbb{Z}_n^\times = \{0, 1, 2, \ldots, p-1\}$. This group is cyclic, i.e., $\forall a \in \mathbb{Z}_p^\times \exists g \in \mathbb{N}_{p-2}^0 : a = g^i = g^{-j},\ 0 \leq i, j \leq p-2$, where g is known as the *primitive root modulo* p and $\mathbb{N}_{p-2}^0 = \{0, 1, 2, \ldots, p-2\}$. In practice, there is no general formula for finding g for p, but in this implementation N is known and therefore g may be stored in a lookup table.

The general form of the DFT of length p is given by
Equation:

$$X_k = \sum_{n=0}^{p-1} x_n \omega_p^{nk} = \widetilde{x} + \sum_{n=1}^{p-1} x_n \omega_p^{nk}, \quad k = 1, 2, \ldots, p-1$$

Since the twiddle factor, ω_p^{nk}, is naturally computed modulo p and both n and k range from $1, 2, \ldots, p-1$, we can rewrite the above formula using g :
Equation:

$$X_{g^{-r}} = \widetilde{x} + \sum_{q=0}^{p-2} x_{g^q} \omega_p^{g^{-(r-q)}}, \quad r = 0, 1, \ldots, p-2$$

This formulation is of the form of a cyclic convolution of two sequences α and β where $\alpha_q = h_{g^q}$ and $\beta_q = \omega_p^{g^{-q}}$ of length $p-1$. This convolution is computed via the convolution theorem:
Equation:

$$X - \widetilde{x} = \mathrm{DFT}^{-1}[\mathrm{DFT}[\alpha] \cdot \mathrm{DFT}[\beta]]$$

For speed, the sequence α is zero-padded between its zeroth and first index to a length M which is a power of 2 such that $M \geq 2p-3$ and β is cyclically repeated to be length M. Since N is known, it is possible to store the DFT of the sequence β in a lookup table. The DFT of α is then computed using the standard radix-2 Cooley-Tukey algorithm. The two DFTs are multiplied pointwise and then the inverse DFT is again calculated using the standard Cooley-Tukey algorithm. The first p elements are then the result of the cyclic convolution. The final result is attained after adding back the DC offset.

For small numerical examples demonstrating this implementation of Rader's FFT, see this excellent guide here.

Computational Experiment

Experiment Design

We wanted to determine the most computationally efficient demodulation method for the digital multitone scheme. In our experiment, we compared the operation counts of an optimized FFT meta algorithm, a partial DFT computation (the DFT computed only for the nonzero coefficients), and FFAST in demodulating various message signals. We chose these demodulation methods because they are currently the most efficient methods for DMT demodulation. The experiments were run using MATLAB 2014a. We chose to run our experiment in MATLAB for its rapid prototyping environment.

While we were interested in comparing the computational efficiency of these algorithms, we chose to record operation counts rather than run times. Run times are unreliable metrics on machines with multitasking operating systems, especially when using highly optimized programs like MATLAB. We chose to count complex additions and multiplications as one operation each. We did not count conditional statements as operations because in most general processors, they only require a single cycle. We counted complex exponentials and trigonometric functions as one operation because they may be implemented using lookup tables.

Meta-FFT Algorithm Implementation

The two main goals for our implementation of the optimized FFT meta algorithm were to i) create an algorithm that performs without the need to zero-pad the signal and ii) allows us to count operations. Well-behaved signal lengths in our implementation of the algorithm have the form of $N = n_0 n_1 n_2$, where n_i are coprimes. To exploit this structure, the optimized FFT meta algorithm, implemented in `meta_fft.m`, uses a self-sorting mixed-radix complex FFT [link]. For sub-transforms of length

$2 \leq N \leq 6$, short-length Winograd transforms are applied to conserve operation count. All other transforms are computed using Rader's FFT algorithm [link].

FFAST Implementation

The FFAST algorithm implementation requires the signal itself, the length of the signal, and a vector of the downsampling coefficients.

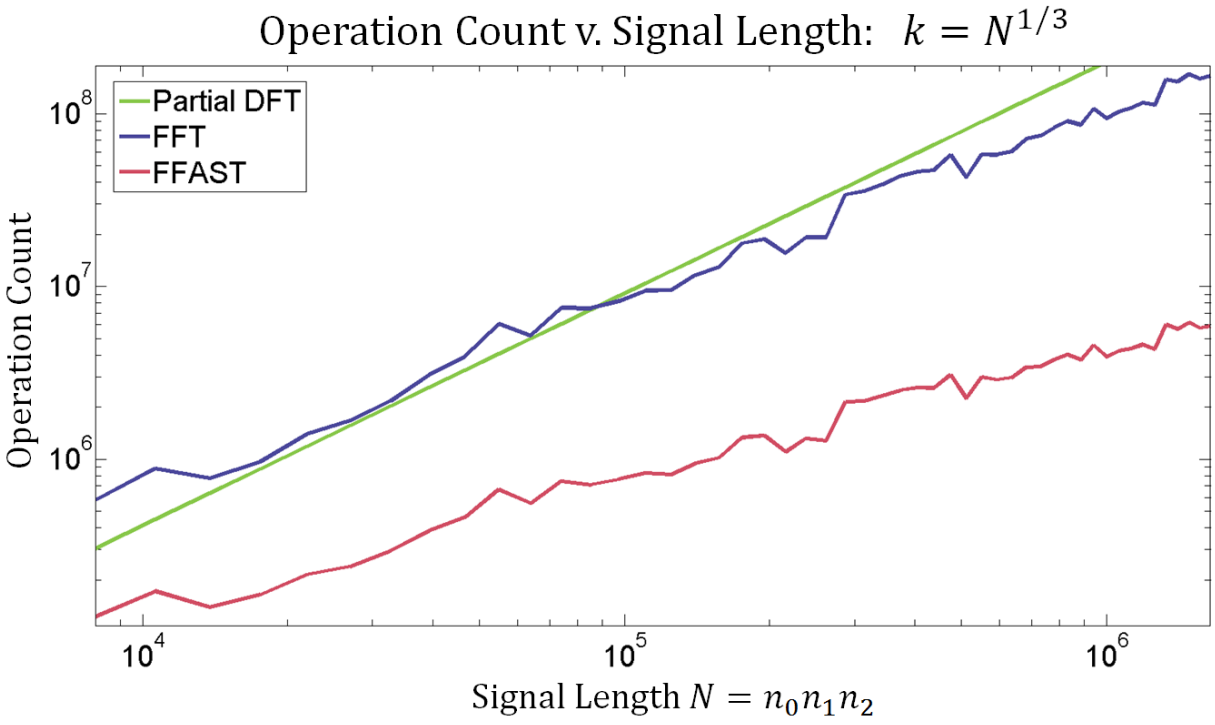
The FFAST algorithm was implemented in two files. The first file, `ffast_front_end.m`, downsamples the signal by each of the coprimes and feeds shifted and unshifted versions of the downsampling to the `meta_fft.m` file, to get operation counts and the relevant FFTs. Once the relevant DFT pairs are generated, `ffast_front_end.m` calls the back end of the algorithm.

The second file, `peeling_decoder.m`, implements the peeling module of FFAST to backsolve the bipartite graph. The program will return a flag if the algorithm encounters no singletons at a stage where it has not been fully solved.

Numerical Results

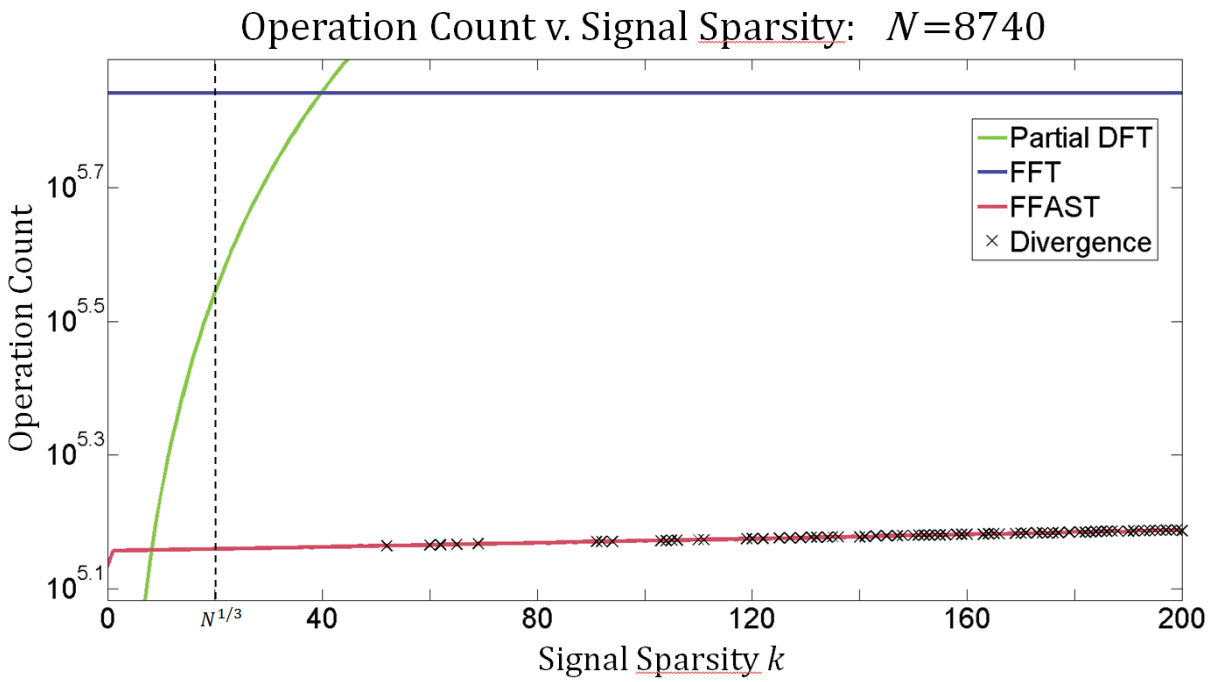
In our first experiment, we varied the signal length N and observed the operation counts required for the optimized FFT meta algorithm, the partial DFT, and FFAST. We constructed each signal in the Fourier domain by randomly selecting k values from the set of integers $\{1, \cdots, N\}$ and setting the corresponding DFT coefficients to $\frac{k}{2}$. For each signal, we put $k = N^{1/3}$, which is the greatest allowed sparsity in our scheme. The choice of k and values of the k nonzero DFT coefficients are consistent with the DMT scheme. We used MATLAB's library function `ifft()` to compute the corresponding signal and counted the number of operations it took each algorithm to compute the DFT. We observed that the operation count required for FFAST was usually an order of magnitude less that the

operation counts required for both the meta FFT algorithm and the partial DFT. See Fig [link] for the results of the first experiment.



Experiment 1

In our second experiment, we varied the signal sparsity k and observed the operation count required for the optimized FFT meta algorithm, the partial DFT, and FFAST. For this experiment, signal length $N = 8740$ and the signals were constructed in the Fourier domain, as before. We observed an 80% computational decrease from the optimized FFT to FFAST for all $k < N^{\frac{1}{3}}$. However, for $k > N^{\frac{1}{3}}$, we observed signals for which FFAST did not converge. See Fig [link] for the results of the second experiment.



Experiment 2

Note that the operation count for the partial DFT is less than the operation count for FFAST for $k < 8$; this is somewhat misleading because the way that the partial DFT is computed is slightly optimistic. The partial DFT computes only the nonzero coefficients, which are known a priori in this experiment. In the general framework of DMT, one would need to compute all possibly nonzero DFT coefficients, resulting in an operation count higher than that of FFAST.

Discussion and Future Work

In any digital communication scheme there exist design parameters that are independent of the scheme itself: digital sampling rate and system baud rate. These parameters directly determine the values of and in the DMT communication scheme. It is therefore plausible to have many different values of and .

From the results, the FFAST algorithm outperformed the mixed-radix FFT for all signals with lengths greater than , with the additional condition that ¯. Thus, if a communication scheme takes at least samples in the time it takes to send — ¯ simultaneous bits, a sparse FFT will require fewer computations than the tested existing frequency domain schemes, reducing receiver bottlenecking, and will therefore be practical to use with some system designs.

Ultimately, the best way to address the viability of the sparse FFT (and therefore expand on the goals of this paper) is to physically implement a communications system compatible with the algorithm itself. While this paper has attempted to address concerns about the possibility of implementation there are still further matters to consider before a physical interpretation of this algorithm can arise.

The first and foremost matter to consider is that the version of the FFAST algorithm that we implemented only works when the signal is exactly sparse. Practically, the communications scheme would have to work with a noisy channel. A noisy version of the FFAST algorithm does exist [link], however, and should be tested to verify our results in a noisy case.

Second, it would be useful to devise a more efficient communication scheme that takes into consideration the fact that the sparse FFT converges even though it does not“know” where the signal is not sparse. In our experiment, we allotted the first “slots” of the frequency domain of our signal to the sinusoids, a way to guarantee that the frequency sparsity of our

signal would not exceed . This does not take into consideration that for any given there are

Equation:

—

different ways to have a sparse signal of density . Finding a coherent way of organizing these different possibilities and using them will give transmitted signals a much higher density and also allow for a higher baud rate of the system (in the example above, would be increased from 12 bits to 127 simultaneous bits!).

Ultimately, once these considerations are taken into account, a coherent sparse communication system seems much more plausible.

The Team

Christopher Harshaw implemented a GUI for the demo of the project, JJ Allred researched and coded the meta-FFT algorithm implementation, and 42 Prieto coded the FFAST front and back ends. All members worked on the report and poster. You can download our poster as a pdf here. . Team Awesome, signing off.